Đã bắt đầu vào lúc	Thứ sáu, 2 Tháng mười hai 2022, 11:38 PM
Tình trạng	Đã hoàn thành
Hoàn thành vào lúc	Thứ sáu, 2 Tháng mười hai 2022, 11:49 PM
Thời gian thực hiện	10 phút 52 giây
Điểm	7,90/8,00
Điểm	**9,88** của 10,00 (**98,75**%)

Câu hỏi 1

Chính xác

Điểm 1,00 của 1,00

Implement functions: **Peek, Pop, Size, Empty, Contains** to a maxHeap. If the function cannot execute, **return -1**.

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>
#include <cmath>
#include <vector>
#include <algorithm>
using namespace std;
#define SEPARATOR "#<ab@17943918#@>#"
template<class T>
class Heap {
protected:
    T* elements;
    int capacity;
    int count;
public:
    Heap()
    {
        this->capacity = 10;
        this->count = 0;
        this->elements = new T[capacity];
    }
    ~Heap()
    {
        delete[]elements;
    }
    void push(T item);

    bool isEmpty();
    bool contains(T item);
    T peek();
    bool pop();
    int size();

    void printHeap()
    {
        cout << "Max Heap [ ";
        for (int i = 0; i < count; i++)
            cout << elements[i] << " ";
        cout << "]\n";
    }
private:
    void ensureCapacity(int minCapacity);
    void reheapUp(int position);
    void reheapDown(int position);
};
//Your code goes here
```


For example:

Test	Result

Heap<int> maxHeap;
for (int i=0;i<10;i++){
maxHeap.push(i);
}
cout << maxHeap.size();
	10

Test	Result

Heap<int> maxHeap;
for (int i=0;i<10;i++){
maxHeap.push(i);
}
cout << maxHeap.isEmpty();
	0

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
template<class T>
int Heap<T>::size(){
    return count;
}

template<class T>
bool Heap<T>::isEmpty(){
    if(count) return false;
    else return true;
}

template<class T>
T Heap<T>::peek(){
    return elements[0];
}

template<class T>
bool Heap<T>::contains(T item){
    for(int i = 0; i < count; i++){
        if(item == elements[i]) return true;
    }
    return false;
}

template<class T>
bool Heap<T>::pop(){
    if(count == 0) return false;
    elements[0] = elements[count-1];
    //delete (elements+count-1);
    reheapDown(0);
    count--;
    return true;
}
```

	Test	Expected	Got	

✔	`Heap<int> maxHeap;`
	`for (int i=0;i<10;i++){`
	`  maxHeap.push(i);`
	`}`
	`cout << maxHeap.size();`
		10	10	✔

✔	`Heap<int> maxHeap;`
	`for (int i=0;i<10;i++){`
	`  maxHeap.push(i);`
	`}`
	`cout << maxHeap.isEmpty();`
		0	0	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 2

Chính xác

Điểm 1,00 của 1,00

Implement function push to push a new item to a maxHeap. You also have to implement ensureCapacity and reheapUp to help you achieve that.

```
template
class Heap{
protected:
    T *elements;
    int capacity;
    int count;

public:
    Heap()
    {
        this->capacity = 10;
        this->count = 0;
        this->elements = new T[capacity];
    }
    ~Heap()
    {
        delete []elements;
    }
    void push(T item);
    void printHeap()
    {
        cout << "Max Heap [ ";
        for (int i = 0; i < count; i++)
            cout << elements[i] << " ";
        cout << "]";
    }

private:
    void ensureCapacity(int minCapacity);
    void reheapUp(int position);
};

// Your code here
```


For example:

Test	Result

`Heap<int> maxHeap;`
`for(int i = 0; i <5;i++)`
`  maxHeap.push(i);`
`maxHeap.printHeap();`
	`Max Heap [ 4 3 1 0 2 ]`

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
1  template<class T>
2  void Heap<T>::push(T item){
3      count++;
4      ensureCapacity(capacity);
5      elements[count-1] = item;
6      reheapUp(count-1);
7  }
8
```

```cpp
template<class T>
void Heap<T>::ensureCapacity(int minCapacity){
    if(count > capacity) capacity = minCapacity + 1;
    T* newE = new T[capacity];
    for(int i = 0; i < count-1; i ++) newE[i] = elements[i];
    delete[] elements;
    elements = newE;
}

template<class T>
void Heap<T>::reheapUp(int position){
    if(elements[position] > elements[(position-1)/2]){
        swap(elements[position],elements[(position-1)/2]);
        reheapUp((position-1)/2);
    }
}
```


	Test	Expected	Got	

✔	`Heap<int> maxHeap;`
	`for(int i = 0; i <5;i++)`
	`maxHeap.push(i);`
	`maxHeap.printHeap();`
		`Max Heap [ 4 3 1 0 2 ]`	`Max Heap [ 4 3 1 0 2 ]`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 3

Chính xác

Điểm 1,00 của 1,00

Given an array which the elements in it are random. Now we want to build a Max heap from this array. Implement functions Reheap up and Reheap down to heapify element at index position. We will use it to build a heap in next question.

To keep things simple, this question will separate the heap array, not store it in the class heap

```
void reheapDown(int maxHeap[], int numberOfElements, int index);
void reheapUp(int maxHeap[], int numberOfElements, int index);
```

For example:

Test	Result

int arr[] = {1,2,3,4,5,6,7,8};
int size = sizeof(arr)/sizeof(arr[0]);
reheapDown(arr,size,0);
cout << "[";
for(int i=0;i<size;i++)
cout << arr[i] << " ";
cout << "]";
	[3 2 7 4 5 6 1 8]

int arr[] = {1,2,3,4,5,6,7,8};
int size = sizeof(arr)/sizeof(arr[0]);
reheapUp(arr,size,7);
cout << "[";
for(int i=0;i<size;i++)
cout << arr[i] << " ";
cout << "]";
	[8 1 3 2 5 6 7 4]

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
1  void reheapDown(int maxHeap[], int numberOfElements, int index)
2  {
3      int childLeft = 2*index+1;
4      int childRight = 2*index+2;
5      // has left and right
6      if(childRight < numberOfElements){
7          int maxChild = max(maxHeap[childLeft], maxHeap[childRight]);
8          int indexMaxChild = (maxChild == maxHeap[childLeft])? childLeft : childRight;
9          if(maxHeap[index] < maxChild){
10             swap(maxHeap[index],maxHeap[indexMaxChild]);
11             reheapDown(maxHeap, numberOfElements,indexMaxChild);
12         }
13         return;
14     }
15     //has left
16     if(childLeft < numberOfElements){
17         if(maxHeap[index] < maxHeap[childLeft]){
18             swap(maxHeap[index], maxHeap[childLeft]);
19             return;
20         }
21     }
22     // a leaf
23     return;
24
25 }
26
27 void reheapUp(int maxHeap[], int numberOfElements, int index)
```

```
{
    int par = (index-1)/2;
    //has par
    if(par >= 0){
        if(maxHeap[index] > maxHeap[par]){
            swap(maxHeap[index], maxHeap[par]);
            reheapUp(maxHeap, numberOfElements,par);
        }
    }
    return;

}
```


	Test	Expected	Got	

✔	`int arr[] = {1,2,3,4,5,6,7,8};`
	`int size = sizeof(arr)/sizeof(arr[0]);`
	`reheapDown(arr,size,0);`
	`cout << "[ ";`
	`for(int i=0;i<size;i++)`
	`  cout << arr[i] << " ";`
	`cout << "]";`
		[3 2 7 4 5 6 1 8]	[3 2 7 4 5 6 1 8]	✔

✔	`int arr[] = {1,2,3,4,5,6,7,8};`
	`int size = sizeof(arr)/sizeof(arr[0]);`
	`reheapUp(arr,size,7);`
	`cout << "[ ";`
	`for(int i=0;i<size;i++)`
	`  cout << arr[i] << " ";`
	`cout << "]";`
		[8 1 3 2 5 6 7 4]	[8 1 3 2 5 6 7 4]	✔

✔	`int arr[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};`
	`int size = sizeof(arr)/sizeof(arr[0]);`
	`reheapUp(arr,size,13);`
	`reheapUp(arr,size,12);`
	`cout << "[ ";`
	`for(int i=0;i<size;i++)`
	`  cout << arr[i] << " ";`
	`cout << "]";`
		[14 2 13 4 5 1 3 8 9 10 11 12 6 7 15]	[14 2 13 4 5 1 3 8 9 10 11 12 6 7 15]	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 4

Đúng một phần

Điểm 0,90 của 1,00

Implement method remove to **remove** the element with given value from a **maxHeap**, **clear** to remove all elements and bring the heap back to the initial state. You also have to implement method **getItem** to help you. Some given methods that you don't need to implement again are **push**, **printHeap**, **ensureCapacity**, **reheapUp**, **reheapDown**.

```
class Heap {
protected:
    T* elements;
    int capacity;
    int count;
public:
    Heap()
    {
        this->capacity = 10;
        this->count = 0;
        this->elements = new T[capacity];
    }
    ~Heap()
    {
        delete[]elements;
    }
    void push(T item);
    int getItem(T item);
    void remove(T item);
    void clear();
    void printHeap()
    {
        cout << "Max Heap [ ";
        for (int i = 0; i < count; i++)
            cout << elements[i] << " ";
        cout << "]\n";
    }
private:
    void ensureCapacity(int minCapacity);
    void reheapUp(int position);
    void reheapDown(int position);
};

// Your code here
```


For example:

Test	Result

`Heap<int> maxHeap;`
`int arr[] = {42,35,30,15,20,21,18,3,7,14};`
`for (int i = 0; i < 10; i++)`
`  maxHeap.push(arr[i]);`
`maxHeap.remove(42);`
`maxHeap.remove(35);`
`maxHeap.remove(30);`
`maxHeap.printHeap();`
	`Max Heap [ 21 20 18 15 14 7 3 ]`

`Heap<int> maxHeap;`
`int arr[] = {78, 67, 32, 56, 8, 23, 19, 45};`
`for (int i = 0; i < 8; i++)`
`  maxHeap.push(arr[i]);`
`maxHeap.remove(78);`
`maxHeap.printHeap();`
	`Max Heap [ 67 56 32 45 8 23 19 ]`

Test	Result

Heap<int> maxHeap;
int arr[] = { 13, 19, 20, 7, 15, 12, 16, 10, 8, 9, 3, 6, 18, 2, 14, 1, 17, 4, 11, 5 };
for (int i = 0; i < 20; ++i)
maxHeap.push(arr[i]);
maxHeap.clear();
maxHeap.printHeap();
	Max Heap [ ]

Answer: (penalty regime: 10, 20, 30, ... %)

Reset answer

```
template<class T>
int Heap<T>::getItem(T item) {
    // TODO: return the index of item in heap
    for (int i=0; i<count; i++)
    {
        if (elements[i]==item) return i;
    }
    return -1;
}

template<class T>
void Heap<T>::remove(T item) {
    // TODO: remove the element with value equal to item
    int index = getItem(item);
    if (index<0) return;

    elements[index] = elements[count-1];
    count--;
    reheapDown(index);
}

template<class T>
void Heap<T>::clear() {
    // TODO: delete all elements in heap
    //while (elements!=NULL)
    //{
        //remove(elements[0]);
    //}
    count = 0;
}
```


	Test	Expected	Got	

✔	Heap<int> maxHeap;
	int arr[] = {42,35,30,15,20,21,18,3,7,14};
	for (int i = 0; i < 10; i++)
	maxHeap.push(arr[i]);
	maxHeap.remove(42);
	maxHeap.remove(35);
	maxHeap.remove(30);
	maxHeap.printHeap();
		Max Heap [21 20 18 15 14 7 3]	Max Heap [21 20 18 15 14 7 3]	✔

	Test	Expected	Got	

✔	`Heap<int> maxHeap;`
	`int arr[] = {78, 67, 32, 56, 8, 23, 19, 45};`
	`for (int i = 0; i < 8; i++)`
	`  maxHeap.push(arr[i]);`
	`maxHeap.remove(78);`
	`maxHeap.printHeap();`
		`Max Heap [ 67 56 32 45 8 23 19 ]`	`Max Heap [ 67 56 32 45 8 23 19 ]`	✔

✔	`Heap<int> maxHeap;`
	`int arr[] = { 13, 19, 20, 7, 15, 12, 16, 10, 8, 9, 3, 6, 18, 2, 14, 1, 17, 4, 11, 5 };`
	`for (int i = 0; i < 20; ++i)`
	`  maxHeap.push(arr[i]);`
	`maxHeap.clear();`
	`maxHeap.printHeap();`
		`Max Heap [ ]`	`Max Heap [ ]`	✔

Your code failed one or more hidden tests.

Đúng một phần

Điểm cho bài nộp này: 0,90/1,00.

Câu hỏi 5

Chính xác

Điểm 1,00 của 1,00

Your task is to implement heap sort (in ascending order) on an unsorted array.

```
#define SEPARATOR "#<ab@17943918#@>#"
#ifndef SORTING_H
#define SORTING_H
#include <iostream>
#include <queue>
using namespace std;
template <class T>
class Sorting {
public:
    /* Function to print an array */
    static void printArray(T *start, T *end)
    {
        long size = end - start;
        for (int i = 0; i < size - 1; i++)
            cout << start[i] << ", ";
        cout << start[size - 1];
        cout << endl;
    }

    //Helping functions go here
    static void heapSort(T* start, T* end){
        //TODO
        Sorting<T>::printArray(start,end);
    }

};
#endif /* SORTING_H */
```


For example:

Test	Result
int arr[4]={4,2,9,1}; Sorting<int>::heapSort(&arr[0],&arr[4]);	1, 2, 4, 9
int arr[4]={-1,0,2,3}; Sorting<int>::heapSort(&arr[0],&arr[4]);	-1, 0, 2, 3

Answer: (penalty regime: 0, 0, 5, ... %)

Reset answer

```
1  static void reheapDown(T* start, T* end, int index){
2      int count = end - start;
3      int childLeft = 2*index +1;
4      int childRight = 2*index+2;
5      if(childRight < count){
6          int indexMax = (start[childRight] > start[childLeft])? childRight : childLeft;
7          if(start[index] < start[indexMax]){
8              swap(start[index], start[indexMax]);
9              reheapDown(start,end,indexMax);
10         }
11         return;
12     }
13     if(childLeft < count){
14         if(start[index] < start[childLeft]){
15             swap(start[index],start[childLeft]);
16         }
17         return;
```

```
    }
    return;
}
static void buildHeap(T* start, T* end){
    int lastNonLeaf = (end-start)/2-1;
    for(int i = lastNonLeaf; i>=0;i--){
        reheapDown(start,end,i);
    }

}
static void heapSort(T* start, T* end){
    buildHeap(start,end);
    queue<T> q;
    int count = end-start;
    for(int i = 0; i < end-start; i++){
        q.push(start[0]);
        swap(start[0],start[count-1]);
        reheapDown(start,start+count-1,0);
        count--;
    }
    for(int i = end-start-1; i >= 0; i--){
        start[i] = q.front();
        q.pop();
    }
    Sorting<T>::printArray(start,end);
}
```

	Test	Expected	Got	
✔	`int arr[4]={4,2,9,1};` `Sorting<int>::heapSort(&arr[0],&arr[4]);`	`1, 2, 4, 9`	`1, 2, 4, 9`	✔
✔	`int arr[4]={-1,0,2,3};` `Sorting<int>::heapSort(&arr[0],&arr[4]);`	`-1, 0, 2, 3`	`-1, 0, 2, 3`	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 6

Chính xác

Điểm 1,00 của 1,00

In a fast food restaurant, a customer is served by following the first-come, first-served rule. The manager wants to minimize the total waiting time of his customers. So he gets to decide who is served first, regardless of how sooner or later a person comes.

Different kinds of food take different amounts of time to cook. And he can't cook food for two customers at the same time, which means when he start cooking for customer A, he has to finish A 's order before cooking for customer B. For example, if there are 3 customers and they come at time 0, 1, 2 respectively, the time needed to cook their food is 3, 9, 6 respectively. If the manager uses first-come, first-served rule to serve his customer, the total waiting time will be 3 + 11 + 16 = 30. In case the manager serves his customer in order 1, 3, 2, the total waiting time will be 3 + 7 + 17 = 27.

Note: The manager does not know about the future orders.

In this question, you should implement function **minWaitingTime** to help the customer find minimum total waiting time to serve all his customers. You are also encouraged to use data structure **Heap** to complete this question. You can use your own code of **Heap**, or use functions related to Heap in library <algorithm>.

For example:

Test	Result

`int n = 3;`
`int arrvalTime[] = { 0, 1, 2 };`
`int completeTime[] = { 3, 9, 6 };`
 	
`cout << minWaitingTime(n, arrvalTime, completeTime);`
	27

`int n = 4;`
`int arrvalTime[] = { 0, 4, 2, 5 };`
`int completeTime[] = { 4, 2, 3, 4 };`
 	
`cout << minWaitingTime(n, arrvalTime, completeTime);`
	21

Answer: (penalty regime: 10, 20, 30, ... %)

Reset answer

```
class Customer{
public:
    int arrvalTime;
    int completeTime;
    Customer(int arrvalTime,int completeTime):
        arrvalTime(arrvalTime),completeTime(completeTime){};
    Customer(){};
};
void sort(int n, int* arrvalTime, int* completeTime){
    int current = 0;
    bool flag = false;
    while(current < n && !flag){
        flag = true;
        int i = n-1;
        while(i> current){
            if(arrvalTime[i] < arrvalTime[i-1]){
                swap(arrvalTime[i],arrvalTime[i-1]);
                swap(completeTime[i],completeTime[i-1]);
                flag = false;
            }
            else if(arrvalTime[i] == arrvalTime[i-1]){
                if(completeTime[i] < completeTime[i-1]){
```

	Test	Expected	Got	

✔	`int n = 3;`
	`int arrvalTime[] = { 0, 1, 2 };`
	`int completeTime[] = { 3, 9, 6 };`
	
	`cout << minWaitingTime(n, arrvalTime, completeTime);`
		27	27	✔

✔	`int n = 4;`
	`int arrvalTime[] = { 0, 4, 2, 5 };`
	`int completeTime[] = { 4, 2, 3, 4 };`
	
	`cout << minWaitingTime(n, arrvalTime, completeTime);`
		21	21	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 7

Chính xác

Điểm 1,00 của 1,00

Cho template của class PrinterQueue có 2 phương thức bắt buộc:

1. `addNewRequest(int priority, string fileName)`

Phương thức đầu tiên sẽ thêm 1 file vào danh sách hàng đợi của máy in (bao gồm độ ưu tiên và tên file). Test case sẽ có tối đa 100 file cùng lúc trong hàng đợi

2. `print()`

Phương thức thứ hai sẽ in tên file kèm xuống dòng và xóa nó ra khỏi hàng đợi. Nếu không có file nào trong hàng đợi, phương thức sẽ in ra "No file to print" kèm xuống dòng.

PrinterQueue tuân theo các quy tắc sau:

- fileName có độ ưu tiên cao nhất sẽ được in trước.
- Các fileName có cùng độ ưu tiên sẽ in theo thứ tự FIFO (First In First Out) order.

Nhiệm vụ của bạn là hiện thực class PrinterQueue thỏa mãn các yêu cầu dữ liệu trên

Lưu ý: Bạn có thể thay đổi mọi thứ, thêm thư viện cần thiết ngoại trừ thay đổi tên class, prototype của 2 public method bắt buộc.

Giải thích testcase 1: File goodbye.pdf có độ ưu tiên là 2 và được thêm vào sớm hơn file goodnight.pdf (độ ưu tiên = 2) nên sẽ được in trước, sau đó đến file goodnight.pdf và cuối cùng là hello.pdf có độ ưu tiên thấp nhất (1)

For example:

Test	Result

`PrinterQueue* myPrinterQueue = new PrinterQueue();`
`myPrinterQueue->addNewRequest(1, "hello.pdf");`
`myPrinterQueue->addNewRequest(2, "goodbye.pdf");`
`myPrinterQueue->addNewRequest(2, "goodnight.pdf");`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
	`goodbye.pdf`
	`goodnight.pdf`
	`hello.pdf`

`PrinterQueue* myPrinterQueue = new PrinterQueue();`
`myPrinterQueue->addNewRequest(1, "hello.pdf");`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
`myPrinterQueue->print();`
	`hello.pdf`
	`No file to print`
	`No file to print`

Answer: (penalty regime: 0, 0, 0, 100 %)

Reset answer

```
#include <cstdlib>
#include <string>
class PrinterQueue
{
    class PriorityString {
    public:
        string fileName;
        int key;
        PriorityString(int priority, int index, string fileName) {
            this->fileName = fileName;
            string q = to_string(100 - index);
            while (int(q.length()) != 3) q = "0" + q;
            this->key = stoi(to_string(priority) + q);
        }
        PriorityString() {};
    };
    PriorityString arr[100];
```

```
18      int count;
19      int size;
20  public:
21      PrinterQueue() {
22          count = 0;
23          //arr = new PriorityString*[100];
24          size = 0;
25      }
26      void reheapDown(int index) {
27          int childL = 2 * index + 1;
28          int childR = 2 * index + 2;
29          if (childR < size) {
30              int indexMax = (arr[childL].key > arr[childR].key) ? childL : childR;
31              PriorityString tmp = arr[index];
32              if (arr[index].key < arr[indexMax].key) {
33                  arr[index] = arr[indexMax];
34                  arr[indexMax] = tmp;
35                  reheapDown(indexMax);
36
37              }
38              return;
39          }
40          if (childL < size) {
41              int indexMax = childL;
42              PriorityString tmp = arr[index];
43              if (arr[index].key < arr[indexMax].key) {
44                  arr[index] = arr[indexMax];
45                  arr[indexMax] = tmp;
46                  reheapDown(indexMax);
47
48              }
49              return;
```

	Test	Expected	Got	

✔	PrinterQueue* myPrinterQueue = new PrinterQueue();
	myPrinterQueue->addNewRequest(1, "hello.pdf");
	myPrinterQueue->addNewRequest(2, "goodbye.pdf");
	myPrinterQueue->addNewRequest(2, "goodnight.pdf");
	myPrinterQueue->print();
	myPrinterQueue->print();
	myPrinterQueue->print();
		goodbye.pdf
		goodnight.pdf
		hello.pdf
			goodbye.pdf
			goodnight.pdf
			hello.pdf
				✔

✔	PrinterQueue* myPrinterQueue = new PrinterQueue();
	myPrinterQueue->addNewRequest(1, "hello.pdf");
	myPrinterQueue->print();
	myPrinterQueue->print();
	myPrinterQueue->print();
		hello.pdf
		No file to print
		No file to print
			hello.pdf
			No file to print
			No file to print
				✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.

Câu hỏi 8

Chính xác

Điểm 1,00 của 1,00

Given an array of non-negative integers. Each time, we can take the smallest integer out of the array, multiply it by 2, and push it back to the array.

Request: Implement function:

```
int leastAfter(vector<int>& nums, int k);
```

Where `nums` is the given array (the length of the array is between 1 and 100000). This function returns the smallest integer in the array after performing the operation `k` times (`k` is between 1 and 100000).

Example:

Given `nums = [2, 3, 5, 7]`.

In the 1st operation, we take `2` out and push back `4`. The array is now `nums = [3, 4, 5, 7]`.

In the 2nd operation, we take `3` out and push back `6`. The array is now `nums = [4, 5, 6, 7]`.

In the 3rd operation, we take `4` out and push back `8`. The array is now `nums = [5, 6, 7, 8]`.

With `k = 3`, the result would be `5`.

Note:

In this exercise, the libraries `iostream, string, cstring, climits, utility, vector, list, stack, queue, map, unordered_map, set, unordered_set, functional, algorithm` has been included and `namespace std` are used. You can write helper functions and classes. Importing other libraries is allowed, but not encouraged, and may result in unexpected errors.

For example:

Test	Result

`vector<int> nums {2, 3, 5, 7};`
`int k = 3;`
`cout << leastAfter(nums, k);`
	5

Answer: (penalty regime: 0, 0, 0, 5, 10, ... %)

Reset answer

```
void reheapUp(vector<int>& nums, int index, int size){
    if(index == 0) return;
    int par = (index-1)/2 ;
    if(nums[index] < nums[par]){
        swap(nums[index], nums[par]);
        reheapUp(nums,par,size);
    }
}
void reheapDown(vector<int>& nums, int index, int size){
    int childL = 2*index+1;
    int childR = 2*index+2;
    if(childR < size){
        int childMin = (nums[childL] < nums[childR])? childL : childR;
        if(nums[index] > nums[childMin]){
            swap(nums[index],nums[childMin]);
            reheapDown(nums,childMin,size);
        }
        return;
    }
    if(childL < size){
        if(nums[index] > nums[childL]){
            swap(nums[index],nums[childL]);
```

	Test	Expected	Got	

✔	`vector<int> nums {2, 3, 5, 7};`
	`int k = 3;`
	`cout << leastAfter(nums, k);`
		5	5	✔

Passed all tests! ✔

Chính xác

Điểm cho bài nộp này: 1,00/1,00.